Parameterized Communicating Automata: Complementation and Model Checking

Benedikt Bollig, Paul Gastin, Akshay Kumar

Parameterized Communicating Automata: Complementation and Model Checking

Benedikt Bollig[1], Paul Gastin[1], and Akshay Kumar[2]

1 **LSV, ENS Cachan & CNRS, France**
{bollig,gastin}@lsv.ens-cachan.fr
2 **Indian Institute of Technology Kanpur, India**
kakshay@iitk.ac.in

Abstract

We study the language-theoretical aspects of parameterized communicating automata (PCAs), in which processes communicate via rendez-vous. A given PCA can be run on any topology of bounded degree such as pipelines, rings, ranked trees, bus topologies, and grids. We show that, under a context bound, which restricts the local behavior of each process, PCAs are effectively complementable. Complementability is considered a key aspect of robust automata models and can, in particular, be exploited for verification. In this paper, we use it to obtain a characterization of context-bounded PCAs in terms of monadic second-order (MSO) logic. As the emptiness problem for context-bounded PCAs is decidable for the classes of pipelines, rings, and trees, their model-checking problem wrt. MSO properties also becomes decidable. While previous work on model checking parameterized systems typically uses temporal logics without next operator, our MSO logic allows one to express several natural next modalities.

1 Introduction

The "regularity" of an automata model is intrinsically tied to characterizations in algebraic or logical formalisms, and to related properties such as closure under complementation and decidability of the emptiness problem. Most notably, the robustness of finite automata is witnessed by the Büchi-Elgot-Trakhtenbrot theorem, stating their expressive equivalence to monadic second-order (MSO) logic. In the past few years, this fundamental result has been extended to models of concurrent systems such as communicating finite-state machines (see [10] for an overview) and multi-pushdown automata (e.g., [11, 12]). Hereby, the system topology, which provides a set of processes and links between them, is usually supposed to be static and fixed in advance. However, in areas such as mobile computing or ad-hoc networks, it is more appropriate to design a program, and guarantee its correctness, independently of the underlying topology, so that the latter becomes a parameter of the system.

There has been a large body of literature on parameterized concurrent systems [1, 2, 6, 8, 9], with a focus on verification: Does the given system satisfy a specification independently of the number of processes? A variety of different models have been introduced, covering a wide range of communication paradigms such as broadcasting, rendez-vous, token-passing, etc. So far, however, it is fair to say that there is no such thing as a canonical or "robust" model of parameterized concurrent systems.

Parameterized Communicating Automata. This paper tries to take a step forward towards such a model. It is in line with a study of a *language theory* of parameterized concurrent systems that has been initiated in [3,4]. We resume the model of parameterized communicating automata (PCAs), a conservative extension of classical communicating finite-state machines [5]. While the latter run a fixed set of processes, a PCA can be run on *any* topology of bounded degree, such as pipelines, rings, ranked trees, bus topologies, or grids. A topology is

a graph, whose nodes represent processes that are connected via interfaces. Every process will run a local automaton executing send and receive actions, which allows it to communicate with an adjacent process in a rendez-vous fashion. As we are interested in language-theoretical properties, we associate, with a given PCA, the set of all possible executions. An execution includes the underlying topology, the events that each process executes, and the causal dependencies that exist between events. This language-theoretic view is different from most previous approaches to parameterized concurrent systems, which rather consider the transition system of reachable configurations. Yet, it will finally allow us to study such important concepts like complementation and monadic second-order (MSO) logic. Note that logical characterizations of PCAs have been obtained in [3]. However, those logics use negation in a restricted way, since PCAs are in general not complementable. This asks for restrictions of PCAs that give rise to a *robust* automata model. In this paper, we will therefore impose a bound on the number of *contexts* that each process traverses.

Context Bounds. The efficiency of distributed algorithms and protocols is usually measured in terms of two parameters: the number n of processes, and the number k of contexts. Here, a context, sometimes referred to as *round*, restricts communication of a process to patterns such as "send a message to each neighbor and receive a message from each neighbor". In this paper, we consider more relaxed definitions where, in every context, a process may perform an unbounded number of actions. In an *interface*-context, a process can send and receive an arbitrary number of messages to/from a *fixed* neighbor. A second context-type definition allows for arbitrarily many sends to all neighbors, or receptions from a fixed neighbor.

In general, basic questions such as reachability are undecidable for PCAs, even when we restrict to simple classes of topologies such as pipelines. To get decidability, it is therefore natural to bound one of the aforementioned parameters, n or k. Bounding the number n of processes is known as *cut-off*. However, the trade-off between n and k is often in favor of an up to exponentially smaller k. Moreover, many distributed protocols actually restrict to a bounded number of contexts, such as P2P protocols and certain leader-election protocols. Therefore, bounding the parameter k seems to be an appropriate way to overcome the theoretical limitations of formally verifying parameterized concurrent systems.

Contribution. The most basic verification question of context-bounded PCAs has been considered in [4]: Is there a topology that allows for an accepting run of the given PCA? In the present paper, we go beyond such nonemptiness/reachability issues and consider PCAs as language acceptors. We will show that, under suitable context bounds, PCAs form a robust automata model that is closed under complementation. Complementability relies on a disambiguation construction, which is the key technical contribution of the paper.

Our complementation result has wider applications and implications. In particular, we obtain a characterization of context-bounded PCAs in terms of a monadic second-order logic. Together with the results from [4], this implies that context-bounded model checking of PCAs against MSO logic is decidable for the classes of pipelines, rings, and trees. Note that MSO logic is quite powerful and, unlike in [2, 7], we are not constrained to drop any (next) modality. Actually, a variety of natural next modalities can be expressed in MSO logic, such as process successor, message successor, next event on a neighboring process, etc.

Context-bounds were originally introduced for (sequential) multi-pushdown automata as models of multi-threaded recursive programs [15]. Interestingly, determinization procedures have been used to obtain complementability and MSO characterizations for context-bounded multi-pushdown automata [11, 12]. A pattern that we share with these approaches is that of computing *summaries* in a deterministic way. Overall, however, we have to use quite different techniques, which is due to the fact that, in our model, processes evolve asynchronously.

Outline. In Section 2, we settle some basic notions such as topologies and message sequence charts, which describe the behavior of a system. PCAs and their restrictions are introduced in Section 3. Section 4 presents our main technical contribution: We show that context-bounded PCAs are complementable. This result is exploited in Section 5 to obtain a logical characterization of PCAs and decidability of the model-checking problem wrt. MSO logic. We conclude in Section 6. Missing proof details can be found in the appendix.

2 Preliminaries

For $n \in \mathbb{N}$, we set $[n] := \{1, \dots, n\}$. Let $\mathbb{A}$ be an alphabet and I be an index set. Given a tuple $\bar{a} = (a_i)_{i \in I} \in \mathbb{A}^I$ and $i \in I$, we write $\bar{a}_i$ to denote a_i.

Topologies. We will model concurrent systems without any assumption on the number of processes. However, we will have in mind that processes are arranged in a certain way, for example as pipelines or rings. Once such a class and the number of processes are fixed, we obtain a topology. Formally, a topology is a graph. Its nodes represent processes, which are connected via interfaces. Let $\mathcal{N} = \{a, b, c, \dots\}$ be a *fixed* nonempty finite set of *interface names* (or, simply, *interfaces*). When we consider pipelines or rings, then $\mathcal{N} = \{a, b\}$ where a refers to the right neighbor and b to the left neighbor of a process, respectively. For grids, we will need two more names, which refer to adjacent processes above and below. Ranked trees require an interface for each of the (boundedly many) children of a process, as well as a pointer to the father process. As $\mathcal{N}$ is fixed, topologies are structures of bounded degree.

▶ **Definition 1.** A *topology* over $\mathcal{N}$ is a pair $\mathcal{T} = (P, \longmapsto)$ where P is the nonempty finite set of *processes* and $\longmapsto \subseteq P \times \mathcal{N} \times \mathcal{N} \times P$ is the *edge relation*. We write $p \overset{a\ b}{\longmapsto} q$ for $(p, a, b, q) \in \longmapsto$, which signifies that the a-interface of p points to q, and the b-interface of q points to p. We require that, whenever $p \overset{a\ b}{\longmapsto} q$, the following hold:

(a) $p \neq q$ (there are no self loops),
(b) $q \overset{b\ a}{\longmapsto} p$ (adjacent processes are mutually connected), and
(c) for all $a', b' \in \mathcal{N}$ and $q' \in P$ such that $p \overset{a'\ b'}{\longmapsto} q'$, we have $a = a'$ iff $q = q'$ (an interface points to at most one process, and two distinct interfaces point to distinct processes).

We do not distinguish isomorphic topologies.

▶ **Example 2.** Example topologies are depicted in Figures 1 and 2. In Figure 2, five processes are arranged as a *ring*. Formally, a ring is a topology over $\mathcal{N} = \{a, b\}$ of the form $(\{1, \dots, n\}, \longmapsto)$ where $n \geq 3$ and $\longmapsto = \{\, (i, a, b, (i \bmod n) + 1) \mid i \in [n] \} \cup \{((i \bmod n) + 1, b, a, i) \mid i \in [n]\}$. A ring is uniquely given by its number of processes. Moreover, as we do not distinguish isomorphic topologies, it does not have an "initial" process. A *pipeline* is of the form $(\{1, \dots, n\}, \longmapsto)$ where $n \geq 2$ and $\longmapsto = \{\, (i, a, b, i{+}1) \mid i \in [n{-}1] \} \cup \{(i{+}1, b, a, i) \mid i \in [n-1]\}$. Similarly, one can define ranked trees and grids [3]. ◀

MSO Logic over Topologies. The acceptance condition of a parameterized communicating automaton (PCA, as introduced in the next section) will be given in terms of a formula from *monadic second-order (MSO) logic*, which scans the final configuration reached by a PCA: the underlying topology together with the local final states in which the processes terminate. If S is the finite set of such local states, the formula thus defines a set of S-labeled topologies, i.e., structures $(P, \longmapsto, \lambda)$ where $(P, \longmapsto)$ is a topology and $\lambda : P \to S$. The logic $\mathrm{MSO}_t(S)$ is

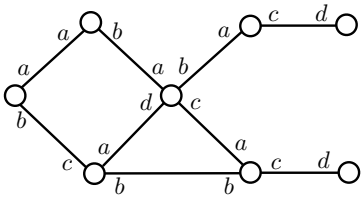


Figure 1 A topology over $\{a, b, c, d\}$

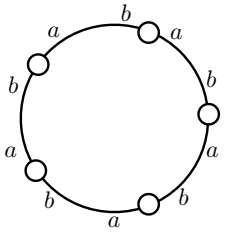


Figure 2 A ring topology

given by the grammar $\mathcal{F} ::= u \overset{a\ b}{\longmapsto} v \mid u = v \mid \lambda(u) = s \mid u \in U \mid \exists u.\mathcal{F} \mid \exists U.\mathcal{F} \mid \neg\mathcal{F} \mid \mathcal{F} \vee \mathcal{F}$ where $a, b \in \mathcal{N}$, $s \in S$, u and v are first-order variables (interpreted as processes), and U is a second-order variable (ranging over sets of processes). Note that we assume an infinite supply of variables. Given a sentence $\mathcal{F} \in \mathrm{MSO}_t(S)$ (i.e., a formula without free variables), we write $L(\mathcal{F})$ for the set of S-labeled topologies $(P, \longmapsto, \lambda)$ that satisfy $\mathcal{F}$. Hereby, satisfaction is defined in the expected manner (cf. also Section 5, presenting an extended logic).

Message Sequence Charts. Recall that our primary concern is a language-theoretic view of parameterized concurrent systems. To this aim, we associate with a system its language, i.e., the set of those behaviors that are generated by an accepting run. One single behavior is given by a message sequence chart (MSC). An MSC consists of a topology (over the given set of interfaces) and a set of events, which reflect the communication actions executed by a system. Events are located on the processes and connected by process and message edges, which reflect causal dependencies (as we consider rendez-vous communication, a message edge has to be interpreted as "simultaneously").

▶ **Definition 3.** A *message sequence chart (MSC)* over $\mathcal{N}$ is a tuple $M = (P, \longmapsto, E, \lhd, \pi)$ where $(P, \longmapsto)$ is a topology, E is the nonempty finite set of *events*, $\lhd \subseteq E \times E$ is the *acyclic* edge relation, which is partitioned into $\lhd_{\mathsf{proc}}$ and $\lhd_{\mathsf{msg}}$, and $\pi : E \to P$ determines the location of an event in the topology; for $p \in P$, we let $E_p := \{e \in E \mid \pi(e) = p\}$. We require that the following hold:

- $\lhd_{\mathsf{proc}}$ is a union $\bigcup_{p \in P} \lhd_p$ where each $\lhd_p \subseteq E_p \times E_p$ is the direct-successor relation of some total order on E_p,
- there is a partition $E = E_! \uplus E_?$ such that $\lhd_{\mathsf{msg}} \subseteq E_! \times E_?$ defines a bijection from $E_!$ to $E_?$,
- for all $(e, f) \in \lhd_{\mathsf{msg}}$, we have $\pi(e) \overset{a\ b}{\longmapsto} \pi(f)$ for some $a, b \in \mathcal{N}$, and
- in the graph $(E, \lhd \cup \lhd_{\mathsf{msg}}^{-1})$, there is no cycle that uses at least one $\lhd_{\mathsf{proc}}$-edge (this ensures rendez-vous communication).

The set of MSCs (over the fixed set $\mathcal{N}$) is denoted by $\mathbb{MSC}$. Like for topologies, we do not distinguish isomorphic MSCs. Let $\Sigma = \{a! \mid a \in \mathcal{N}\} \cup \{a? \mid a \in \mathcal{N}\}$. We define a mapping $\ell_M : E \to \Sigma$ that associates with each event the type of action that it executes: For $(e, f) \in \lhd_{\mathsf{msg}}$ and $a, b \in \mathcal{N}$ such that $\pi(e) \overset{a\ b}{\longmapsto} \pi(f)$, we set $\ell_M(e) = a!$ and $\ell_M(f) = b?$.

▶ **Example 4.** Two example MSCs are depicted in Figure 3, both having the ring with five processes as underlying topology (for the moment, we ignore the state labels s_i of processes). The events are the endpoints of message arrows, which represent $\lhd_{\mathsf{msg}}$. Process edges are implicitly given; they connect successive events located on the same (top-down) process line. Finally, the mapping ℓ_M is illustrated on a few events. ◀

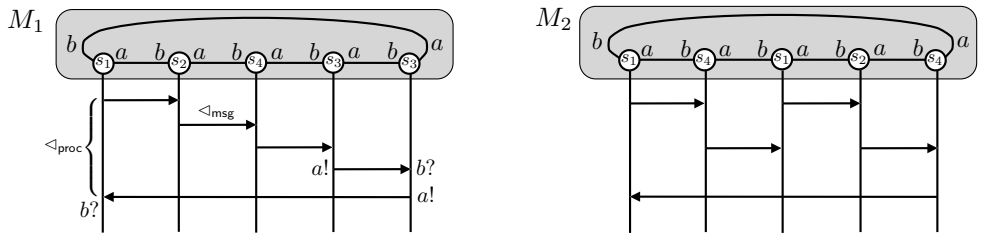


Figure 3 Two MSCs over a ring topology; local states labeling the topology in a PCA run

3 Parameterized Communicating Automata

In this section, we introduce our model of a communicating system that can be run on arbitrary topologies of bounded degree.

The Model and Its Semantics. The idea is that each process of a given topology runs one and the same automaton, whose transitions are labeled with an action of the form $(a!, m)$, which emits a message m through interface a, or $(a?, m)$, which receives m from interface a.

▶ **Definition 5.** A *parameterized communicating automaton (PCA)* over $\mathcal{N}$ is a tuple $\mathcal{A} = (S, \iota, Msg, \Delta, \mathcal{F})$ where S is the finite set of *states*, $\iota \in S$ is the *initial state*, Msg is a nonempty finite set of *messages*, $\Delta \subseteq S \times (\Sigma \times Msg) \times S$, and $\mathcal{F} \in \mathrm{MSO}_t(S)$ is a sentence, representing the acceptance condition.

Let $M = (P, \longmapsto, E, \lhd, \pi)$ be an MSC. A run of $\mathcal{A}$ on M will be a mapping $\rho : E \to S$ satisfying some requirements. Intuitively, $\rho(e)$ is the local state of $\pi(e)$ after executing e. To determine when ρ is a run, we define another mapping, $\rho^- : E \to S$, denoting the source states of a transition: whenever $f \lhd_{\mathsf{proc}} e$, we let $\rho^-(e) = \rho(f)$; moreover, if e is $\lhd_{\mathsf{proc}}$-minimal, we let $\rho^-(e) = \iota$. With this, we say that ρ is a run of $\mathcal{A}$ on M if, for all $(e, f) \in \lhd_{\mathsf{msg}}$, there are $a, b \in \mathcal{N}$ and a message $m \in Msg$ such that $\pi(e) \overset{a\ b}{\longmapsto} \pi(f)$, $(\rho^-(e), (a!, m), \rho(e)) \in \Delta$, and $(\rho^-(f), (b?, m), \rho(f)) \in \Delta$. To determine when ρ is accepting, we collect the last states of all processes and define a mapping $\lambda : P \to S$ as follows. Let $p \in P$. If $E_p = \emptyset$, then $\lambda(p) = \iota$; otherwise, $\lambda(p)$ is set to $\rho(e)$ where e is the unique $\lhd_{\mathsf{proc}}$-maximal event of p. Now, run ρ is *accepting* if $(P, \longmapsto, \lambda) \in L(\mathcal{F})$. The set of MSCs that allow for an accepting run is denoted by $L(\mathcal{A})$.

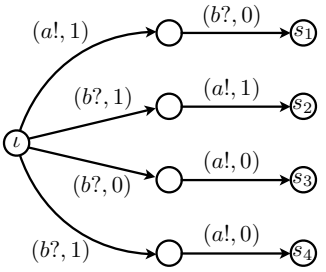
While a run of a PCA is purely operational, it is actually natural to define the acceptance condition in terms of $\mathrm{MSO}_t(S)$, which allows for a global, declarative view of the final configuration. Note that, when we restrict to pipelines, rings, or ranked trees, the acceptance condition could be defined as a finite (tree, respectively) automaton over the alphabet S.

▶ **Example 6.** The PCA from Figure 4 describes a simplified version of the IEEE 802.5 token-ring protocol. For illustration, we consider two different acceptance conditions, $\mathcal{F}$ and $\mathcal{F}'$, giving rise to PCAs $\mathcal{A}_{\text{token}}$ and $\mathcal{A}'_{\text{token}}$, respectively. In both cases, a single binary token, which can carry a value from $m \in \{0, 1\}$, circulates in a ring. Recall that, in a ring topology, every process has an a-neighbor and a b-neighbor (cf. Figure 2). Initially, the token has value 1. A process that has the token may emit a message and pass it along with the token to its a-neighbor. We will abstract the concrete message away and only consider the token value. Whenever a process receives the token from its b-neighbor, it will forward it to its a-neighbor, while (i) leaving the token value unchanged (the process then ends in state s_2 or s_3), or (ii) changing its value from 1 to 0, to signal that the message has been received

(the process then ends in s_4). Once the process that initially launched the token receives the token with value 0, it goes to state s_1.

Note that the acceptance condition $\mathcal{F}$ of $\mathcal{A}_{\text{token}}$ permits those configurations where all processes terminate in one of the states $s_1, \ldots, s_4$. MSC M_1 from Figure 3 depicts an execution of the protocol described above, and we have $M_1 \in L(\mathcal{A}_{\text{token}})$. The state-labelings of processes indicate the final local states that are reached in an accepting run. However, one easily verifies that we also have $M_2 \in L(\mathcal{A}_{\text{token}})$, though M_2 should not be considered as an execution of a token-ring protocol: there are two processes that, independently of each other, emit a message/token and end up in s_1. To model the protocol faithfully and rule out such pathological executions, we change the acceptance condition to $\mathcal{F}'$, which adds the requirement that *exactly* one process terminates in s_1. We actually have $M_1 \in L(\mathcal{A}'_{\text{token}})$ and $M_2 \notin L(\mathcal{A}'_{\text{token}})$. ◀

Note that [3, 4] used weaker acceptance conditions, which cannot access the topology. However, Example 6 shows that an acceptance condition given as an MSO_t-formula offers some flexibility in modeling parameterized systems. For example, it can be used to simulate several process types [4], the idea being that each process runs a local automaton according to its type. All our results go through in this extended setting. Also note that messages (such as the token value in Example 6) could be made apparent in the MSCs. However, we will always need some "hidden" messages, which are common in communicating automata with fixed topology [10] and significantly extend their expressive power.



$$\mathcal{F} \equiv \forall u.\lambda(u) \in \{s_1, \ldots, s_4\}$$
$$\mathcal{F}' \equiv \mathcal{F} \wedge \exists^{=1} u.\lambda(u) = s_1$$

Figure 4 The PCA $\mathcal{A}'_{\text{token}}$

Context-Bounded PCAs. Our main results will rely on a restricted version of PCAs, where every process is constrained to execute a bounded number of *contexts*. As discussed in the introduction, contexts come very naturally when modeling distributed protocols. Actually, the behavior of a single process is often divided into a small, or even bounded, number of *rounds*, each describing some restricted communication pattern. Usually, one considers that a round consists of sending a message to each neighbor followed by receiving a message from each neighbor [13]. In this paper, we consider *contexts*, which are somewhat more general than rounds: In a context, one may potentially execute an unbounded number of actions. Moreover, a round can be simulated by a bounded number of contexts. Actually, there exist several natural definitions. A word $w \in \Sigma^*$ is called an

- ($\mathsf{s}\oplus\mathsf{r}$)-context if $w \in \{a! \mid a \in \mathcal{N}\}^*$ or $w \in \{a? \mid a \in \mathcal{N}\}^*$,
- ($\mathsf{s1}+\mathsf{r1}$)-context if $w \in \{a!, b?\}^*$ for some $a, b \in \mathcal{N}$,
- ($\mathsf{s}\oplus\mathsf{r1}$)-context if $w \in \{a! \mid a \in \mathcal{N}\}^*$ or $w \in \{b?\}^*$ for some $b \in \mathcal{N}$,
- intf-context if $w \in \{a!, a?\}^*$ for some $a \in \mathcal{N}$.

The context type $\mathsf{s1}\oplus\mathsf{r}$ ($w \in \{a!\}^*$ or $w \in \{b? \mid b \in \mathcal{N}\}^*$ for some $a \in \mathcal{N}$) is dual to $\mathsf{s}\oplus\mathsf{r1}$, and we only consider the latter case. All results for $\mathsf{s}\oplus\mathsf{r1}$ in this paper easily transfer to $\mathsf{s1}\oplus\mathsf{r}$.

Let $k \geq 1$ be a natural number and $ct \in \{\mathsf{s}\oplus\mathsf{r}, \mathsf{s1}+\mathsf{r1}, \mathsf{s}\oplus\mathsf{r1}, \mathsf{intf}\}$ be a context type. We say that $w \in \Sigma^*$ is (k, ct)-*bounded* if there are $w_1, \ldots, w_k \in \Sigma^*$ such that $w = w_1 \cdots w_k$ and w_i is a ct-context, for all $i \in [k]$. To lift this definition to MSCs $M = (P, \longmapsto, E, \lhd, \pi)$, we define the projection $M|_p \in \Sigma^*$ of M to a process $p \in P$. Let $e_1 \lhd_{\mathsf{proc}} e_2 \lhd_{\mathsf{proc}} \cdots \lhd_{\mathsf{proc}} e_n$ be the unique process-order preserving enumeration of all events of E_p. We let $M|_p = \ell_M(e_1)\ell_M(e_2)\ldots\ell_M(e_n)$. In particular, $E_p = \emptyset$ implies $M|_p = \varepsilon$. Now, we say that M is (k, ct)-*bounded* if $M|_p$ is (k, ct)-bounded, for all $p \in P$. Let $\mathbb{MSC}_{(k,ct)}$ denote the set

of all (k, ct)-bounded MSCs. Given two sets L and L' of MSCs, we write $L \equiv_{(k,ct)} L'$ if $L \cap \mathbb{MSC}_{(k,ct)} = L' \cap \mathbb{MSC}_{(k,ct)}$.

▶ **Example 7.** Consider the PCAs $\mathcal{A}_{\text{token}}$ and $\mathcal{A}'_{\text{token}}$ from Figure 4. Every process executes at most two events so that we have $L(\mathcal{A}'_{\text{token}}) \subseteq L(\mathcal{A}_{\text{token}}) \subseteq \mathbb{MSC}_{(2,ct)}$ for all context types $ct \in \{\mathsf{s}{\oplus}\mathsf{r}, \mathsf{s1}{+}\mathsf{r1}, \mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$. In particular, the MSCs M_1 and M_2 from Figure 3 are $(2, ct)$-bounded.

4 Context-Bounded PCAs are Complementable

Let $ct \in \{\mathsf{s}{\oplus}\mathsf{r}, \mathsf{s1}{+}\mathsf{r1}, \mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$. We say that PCAs are *ct-complementable* if, for every PCA $\mathcal{A}$ and $k \geq 1$, we can effectively construct a PCA $\mathcal{A}'$ such that $L(\mathcal{A}') \equiv_{(k,ct)} \mathbb{MSC} \setminus L(\mathcal{A})$. In general, PCAs are not complementable, and this even holds under certain context bounds.

▶ **Theorem 8.** *Suppose $\mathcal{N} = \{a, b\}$. For all context types $ct \in \{\mathsf{s}{\oplus}\mathsf{r}, \mathsf{s1}{+}\mathsf{r1}\}$, PCAs are* not *ct-complementable.*

The proof uses results from [14, 16] and can be found in the appendix. However, the situation changes when we move to context types $\mathsf{s}{\oplus}\mathsf{r1}$ and intf. We now present the main result of our paper:

▶ **Theorem 9.** *For all $ct \in \{\mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$, PCAs are ct-complementable.*

The theorem follows directly from a disambiguation construction, which we present as Theorem 10. We call a PCA $\mathcal{A}$ *unambiguous* if, for every MSC M, there is exactly one run (accepting or not) of $\mathcal{A}$ on M. An unambiguous PCA can be easily complemented by negating the acceptance condition.

▶ **Theorem 10.** *Given a PCA $\mathcal{A}$, a natural number $k \geq 1$, and $ct \in \{\mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$, we can effectively construct an unambiguous PCA $\mathcal{A}'$ such that $L(\mathcal{A}) \equiv_{(k,ct)} L(\mathcal{A}')$.*

The PCA from Figure 4 is not unambiguous, since there are runs of $\mathcal{A}_{\text{token}}$ (or $\mathcal{A}'_{\text{token}}$) on the MSC M_1 from Figure 3 ending, for example, in configurations $s_1 s_2 s_4 s_3 s_3$ or $s_1 s_2 s_2 s_4 s_3$. Unfortunately, a simple power-set construction is not applicable to PCAs, due to the hidden message contents. Note that, in the fixed-topology setting, there is a commonly accepted notion of *deterministic* communicating automata [10], which is different from *unambiguous*. We do not know if Theorem 10 holds for deterministic PCAs.

Proof of Theorem 10

In the remainder of this section, we prove Theorem 10. The proof outline is as follows: We first define an intermediate model of *complete deterministic asynchronous automata (CDAAs)*. We will then show that any context-bounded PCA can be converted into a CDAA (Lemma 13) which, in turn, can be converted into an unambiguous PCA (Lemma 12).

▶ **Definition 11.** A *complete deterministic asynchronous automaton (CDAA)* over the set $\mathcal{N}$ is a tuple $\mathcal{B} = (S, \iota, (\delta_{(a,b)})_{(a,b) \in \mathcal{N} \times \mathcal{N}}, \mathcal{F})$ where S, ι, and $\mathcal{F}$ are like in PCAs and, for each $(a, b) \in \mathcal{N} \times \mathcal{N}$, we have a (total) function $\delta_{(a,b)} : (S \times S) \to (S \times S)$.

The main motivation behind introducing CDAAs is that, for a given process p, the functions $(\delta_{(a,b)})_{(a,b) \in \mathcal{N} \times \mathcal{N}}$ can effectively encode the transitions at each of the neighbors of p. Similarly to PCAs, a run of $\mathcal{B}$ on an MSC $M = (P, \longmapsto, E, \lhd, \pi)$ is a mapping $\rho : E \to S$ such that, for all $(e, f) \in \lhd_{\mathsf{msg}}$, there are $a, b \in \mathcal{N}$ satisfying $\pi(e) \overset{a\ b}{\longmapsto} \pi(f)$ and

$\delta_{(a,b)}(\rho^-(e), \rho^-(f)) = (\rho(e), \rho(f))$. Whether a run is accepting or not depends on $\mathcal{F}$ and is defined exactly like in PCAs. The set of MSCs that are accepted by $\mathcal{B}$ is denoted by $L(\mathcal{B})$.

▶ **Lemma 12.** *For every CDAA $\mathcal{B}$, there is an unambiguous PCA $\mathcal{A}$ such that $L(\mathcal{B}) = L(\mathcal{A})$.*

Proof. The idea is that the messages of a PCA "guess" the current state of the receiving process. A message can only be received if the guess is correct, so that the resulting PCA is unambiguous. Let $\mathcal{B} = (S, \iota, (\delta_{(a,b)})_{(a,b)\in\mathcal{N}\times\mathcal{N}}, \mathcal{F})$ be the given CDAA. We let $\mathcal{A} = (S, \iota, Msg, \Delta, \mathcal{F})$ where $Msg = \mathcal{N} \times \mathcal{N} \times S \times S$ and Δ contains, for every transition $\delta_{(a,b)}(s_1, s_2) = (s_1', s_2')$, the tuples $(s_1, a!(a, b, s_1, s_2), s_1')$ and $(s_2, b?(a, b, s_1, s_2), s_2')$. Note that $\mathcal{A}$ is indeed unambiguous. Let $M = (P, \longmapsto, E, \lhd, \pi)$ be an MSC and $\rho : E \to S$. From the run-definitions, we obtain that ρ is an (accepting) run of $\mathcal{B}$ on M iff it is an (accepting, respectively) run of $\mathcal{A}$ on M. It follows that $L(\mathcal{B}) = L(\mathcal{A})$. ◀

Next, we will describe how an arbitrary context-bounded PCA can be transformed into an equivalent CDAA. This construction is our key technical contribution.

▶ **Lemma 13.** *Let $ct \in \{\mathsf{s}{\oplus}\mathsf{r1}, \mathsf{intf}\}$. For every PCA $\mathcal{A}$ and $k \geq 1$, we can effectively construct a CDAA $\mathcal{B}$ such that $L(\mathcal{A}) \equiv_{(k,ct)} L(\mathcal{B})$.*

The remainder of this section is dedicated to the proof of Lemma 13. We do the proof for the more involved case $ct = \mathsf{s}{\oplus}\mathsf{r1}$ and will explain in the appendix what is different if $ct = \mathsf{intf}$. Let $\mathcal{A} = (S, \iota, Msg, \Delta, \mathcal{F})$ be a PCA and $k \geq 1$. In the following, we will construct the required CDAA $\mathcal{B} = (S', \iota', (\delta_{(a,b)})_{(a,b)\in\mathcal{N}\times\mathcal{N}}, \mathcal{F}')$.

The idea behind our construction is that the current sending process simulates the behavior of all its neighboring receiving processes, storing all possible combinations of global source and target states. In Figure 5, in the beginning, p_2 starts sending to p_3 and p_1. Hence p_2 keeps track of the local states at p_1 and p_3 as well. This computation spans over what we call a *zone* (the gray-shaded areas in Figure 5). Whenever a sending (receiving) process changes into a receiving (sending, respectively) process, the role of keeping track of the behavior of neighboring processes gets passed on to the new sending process, which results in a zone switch. We will see that a bounded number of such changes suffice (Lemma 14). Finally, the acceptance condition $\mathcal{F}'$ checks whether the information stored at each of the processes can be coalesced to get a global run of the given PCA $\mathcal{A}$.

Zones. Let $M = (P, \longmapsto, E, \lhd, \pi)$ be an MSC. An *interval* of M is a (possibly empty) subset of E of the form $\{e_1, e_2, \ldots, e_n\}$ such that $e_1 \lhd_{\mathsf{proc}} e_2 \lhd_{\mathsf{proc}} \ldots \lhd_{\mathsf{proc}} e_n$. A *send context* of M is an interval that consists only of send events. A *receive context* of M is an interval $I \subseteq E$ such that there is $a \in \mathcal{N}$ satisfying $\ell_M(e) = a?$ for all $e \in I$. A set $\mathcal{Z} \subseteq E$ is called a *zone* of M if there is a nonempty send context I such that the corresponding receive contexts $I_a = \{f \in E \mid e \lhd_{\mathsf{msg}} f \text{ for some } e \in I \text{ such that } \ell_M(e) = a!\}$ are intervals for all $a \in \mathcal{N}$, and $\mathcal{Z} = I \cup \bigcup_{a\in\mathcal{N}} I_a$.

Zones help us to maintain the *summary* of a possibly unbounded number of messages in a finite space. By Lemma 14, since there is a bound on the number of different zones for each process, the behavior of a PCA can be described succinctly by describing its action on each of the zones.

▶ **Lemma 14.** *[cf. [4]] Let $M = (P, \longmapsto, E, \lhd, \pi)$ be a $(k, \mathsf{s}{\oplus}\mathsf{r1})$-bounded MSC. There is a partitioning of the events of M into zones such that, for each process $p \in P$, the events of E_p belong to at most $K := k \cdot (|\mathcal{N}|^2 + 2|\mathcal{N}| + 1)$ different zones.*

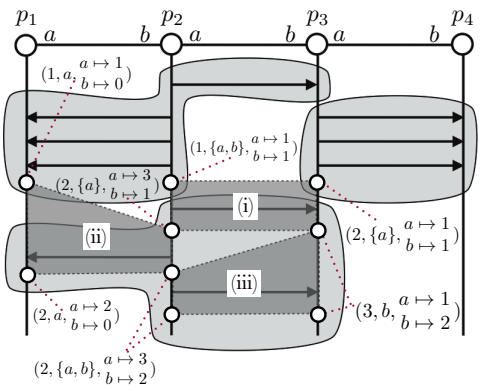


Figure 5 Computing zones in a CDAA $\mathcal{B}$

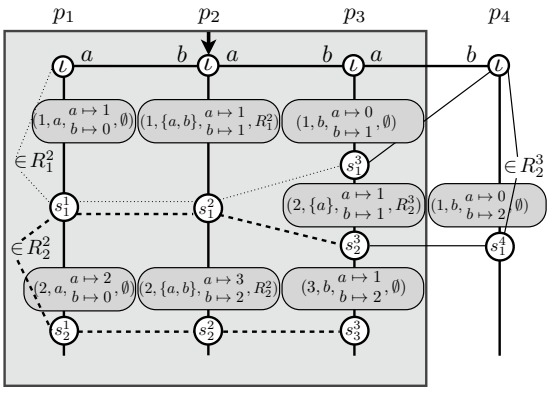


Figure 6 Illustration of $\mathcal{F}' \in \mathrm{MSO}_t(S')$

A CDAA that Computes Zones. We now construct a CDAA that, when running on a $(k, \mathsf{s}{\oplus}\mathsf{r1})$-bounded MSC, computes a "greedy" zone partitioning, for which the bound K from Lemma 14 applies. We explain the intuition by means of Figure 5, which depicts an MSC along with a partitioning of events into different zones. The crucial point for processes is to recognize when to switch to a new zone. Towards this end, a *summary* of the zone is maintained. Each process stores its zone number together with the zone number of its neighboring receiving processes. A sending (receiving) process enters a new zone if the stored zone number of a neighbor does not match the actual zone number of the corresponding neighboring receiving (sending, respectively) process.

In Figure 5, the zone number of p_3 in p_2's first zone is 1. However, at the time of sending the second message from p_2 to p_3, the zone number of p_3 is 2 which does not match the information stored with p_2. This prompts p_2 to define a new zone and update the zone number of p_3.

A *sending process enters a new zone* when (a) it was a receiving process earlier, or (b) the zone number of a receiving process does not match. Similarly, a *receiving process enters a new zone* when (a) it was a sending process earlier, or (b) it was receiving previously from a different process, or (c) the zone number of the sending process does not match. This is formally defined in Equations (1) and (2) below.

We now formally describe the CDAA $\mathcal{B}$. A *zone state* is a tuple (i, τ, κ, R) where

- $i \in \{0, \dots, K\}$ is the current zone number, which indicates that a process traverses its i-th zone (or, equivalently, has switched to a new zone $i-1$ times),
- $\tau \in 2^{\mathcal{N}} \cup \mathcal{N}$ denotes the role of a process in the current zone (if $\tau \subseteq \mathcal{N}$, it has been sending through the interfaces in τ; if $\tau \in \mathcal{N}$, it is receiving from τ),
- $\kappa : \mathcal{N} \to \{0, \dots, K\}$ denotes the knowledge about each neighbor, and
- $R \subseteq (S^{\mathcal{N} \cup \{\mathsf{self}\}})^2$ is the set of possible global steps that the zone may induce; each step involves a source and target state for the current process as well as its neighbors. As the sending process simulates the receivers' steps, we let $R = \emptyset$ whenever $\tau \in \mathcal{N}$.

Let Z be the set of zone states. For $(a, b) \in \mathcal{N} \times \mathcal{N}$, we define a *partial* "update" function $\delta^{\mathrm{zone}}_{(a,b)} : (Z \times Z) \rightharpoonup (Z \times Z)$ by

$$\delta^{\mathrm{zone}}_{(a,b)}((i_1, \tau_1, \kappa_1, R_1), (i_2, \tau_2, \kappa_2, R_2)) = ((i_1', \tau_1', \kappa_1[a \mapsto i_2'], R_1'), (i_2', b, \kappa_2[b \mapsto i_1'], \emptyset))$$

where

$$i_1' = \begin{cases} i_1 + 1 & \text{if } i_1 = 0 \ \text{ or } \ \tau_1 \in \mathcal{N} \ \text{ or } \ (a \in \tau_1 \ \text{ and } \ \kappa_1(a) \neq i_2) \\ i_1 & \text{otherwise} \end{cases} \tag{1}$$

$$i_2' = \begin{cases} i_2 + 1 & \text{if } \tau_2 \neq b \text{ or } \kappa_2(b) \neq i_1 \\ i_2 & \text{otherwise} \end{cases} \tag{2}$$

$$\tau_1' = \begin{cases} \{a\} & \text{if } i_1' = i_1 + 1 \\ \tau_1 \cup \{a\} & \text{otherwise} \end{cases} \qquad R_1' = \begin{cases} R & \text{if } i_1' = i_1 + 1 \\ R_1 \circ R & \text{otherwise} \end{cases} \tag{3}$$

with R being the set of pairs $(\bar{s}, \bar{s}') \in (S^{\mathcal{N} \cup \{\mathsf{self}\}})^2$ such that there is $m \in Msg$ with $(\bar{s}_{\mathsf{self}}, (a!, m), \bar{s}'_{\mathsf{self}}) \in \Delta$, $(\bar{s}_a, (b?, m), \bar{s}'_a) \in \Delta$, and $\bar{s}_c = \bar{s}'_c$ for all $c \in \mathcal{N} \setminus \{a\}$.

The function $\delta^{\text{zone}}_{(a,b)}$ is illustrated in Figure 5 (omitting the R-component) for the three different cases that can occur: (i) both processes increase their zone number; (ii) only the receiver increases its zone number; (iii) none of the processes increases its zone number.

A state of $\mathcal{B}$ is a sequence of zone states, so that a process can keep track of the zones that it traverses. Formally, we let S' be the set of words over Z of the form $(0, \emptyset, \kappa_0, \emptyset)(1, \tau_1, \kappa_1, R_1) \dots (n, \tau_n, \kappa_n, R_n)$ where $n \in \{0, \dots, K\}$ and $\kappa_0(a) = 0$ for all $a \in \mathcal{N}$. The initial state is $\iota' = (0, \emptyset, \kappa_0, \emptyset)$. Note that the size of S' is exponential in K (and, therefore, in k).

We are now ready to define the transition function $\delta_{(a,b)} : (S' \times S') \to (S' \times S')$. Essentially, we take $\delta^{\text{zone}}_{(a,b)}$, but we append a new zone state when the zone number is increased. Let $z_1 = (i_1, \tau_1, \kappa_1, R_1) \in Z$ and $z_2 = (i_2, \tau_2, \kappa_2, R_2) \in Z$. Moreover, suppose $\delta^{\text{zone}}_{(a,b)}(z_1, z_2) = (z_1', z_2')$ where $z_1' = (i_1', \tau_1', \kappa_1', R_1')$ and $z_2' = (i_2', \tau_2', \kappa_2', R_2')$. Then, we let

$$\delta_{(a,b)}(w_1 z_1 \,,\, w_2 z_2) = \begin{cases} (w_1 z_1' \,,\, w_2 z_2') & \text{if } i_1' = i_1 \text{ and } i_2' = i_2 \\ (w_1 z_1' \,,\, w_2 z_2 z_2') & \text{if } i_1' = i_1 \text{ and } i_2' = i_2 + 1 \\ (w_1 z_1 z_1' \,,\, w_2 z_2 z_2') & \text{if } i_1' = i_1 + 1 \text{ and } i_2' = i_2 + 1 \end{cases}$$

Note that the case $i_1' = i_1 + 1 \wedge i_2' = i_2$ can actually never happen. Nonetheless, $\delta_{(a,b)}$ is still a partial function. However, adding a sink state, we easily obtain a function that is complete.

The Acceptance Condition. It remains to determine the acceptance condition of $\mathcal{B}$. The formula $\mathcal{F}' \in \mathrm{MSO}_t$ will check whether there is a concrete choice of local states that is consistent with the zone abstraction and, in particular, with the relations R collected during that run in the zone states. Let T be the set of sequences of the form $\iota s_1 \dots s_n$ where $n \in \{0, \dots, K\}$ and $s_i \in S$ for all i. The idea is that s_i is the local state that a process reaches *after* traversing its i-th zone. The formula will now guess such a sequence for every process and check if this choice matches the abstract run. To verify if the local states correspond to the relation R stored in some constituent sending process p, it is sufficient to look at the adjacent neighbors of p.

This is illustrated in Figure 6 for the zone abstraction from Figure 5. Process p_2, for example, stores both the relations R_1^2 and R_2^2, and we have to check if this corresponds to the sequences from T that the formula had guessed for every process (the white circles). To do so, it is indeed enough to look at the neighborhood of p_2, which is highlighted in gray. The guess is accepted only if the state at the beginning of a zone matches the state at the end of the previous zone. For example, in Figure 6, the formula collects the pair of tuples $((\iota, \iota, \iota), (s_1^1, s_1^2, s_1^3))$ and verifies if it is contained in R_1^2. Also, it collects the pair $((s_1^1, s_1^2, s_2^3), (s_2^1, s_2^2, s_3^3))$ and checks if it is contained in R_2^2. Similarly, looking at the neighborhood of p_3, it verifies whether $((s_1^2, s_1^3, \iota), (s_1^2, s_2^3, s_1^4)) \in R_2^3$.

Let us be more precise. Suppose the final configuration reached by $\mathcal{B}$ is $(P, \longmapsto, \lambda')$ with $\lambda' : P \to S'$. By means of second-order variables U_t, with t ranging over T, the formula

$\mathcal{F}'$ guesses an assignment $\sigma : P \to T$. It will then check that, for all $p \in P$ with, say, $\lambda'(p) = \iota'(1, \tau_1, \kappa_1, R_1) \dots (n_p, \tau_{n_p}, \kappa_{n_p}, R_{n_p}) \in S'$, the following hold:

- the sequence $\sigma(p)$ is of the form $s_0 s_1 \dots s_{n_p}$ (in the following, we let $\sigma(p)_i$ refer to s_i),
- for all $i \in [n_p]$ with $\tau_i \subseteq \mathcal{N}$, there is $(\bar{s}, \bar{s}') \in R_i$ such that (i) $\bar{s}_{\mathsf{self}} = s_{i-1}$ and $\bar{s}'_{\mathsf{self}} = s_i$, and (ii) for all $p \overset{a\ b}{\longmapsto} q$, we have $\bar{s}_a = \sigma(q)_{\kappa_i(a)-1}$ and $\bar{s}'_a = \sigma(q)_{\kappa_i(a)}$ if $a \in \tau_i$, and $\bar{s}_a = \bar{s}'_a = \sigma(q)_{\kappa_i(a)}$ if $a \notin \tau_i$.

These requirements can be expressed in MSO_t. Finally, to incorporate the acceptance condition $\mathcal{F} \in \mathrm{MSO}_t(S)$, we simply replace an atomic formula $\lambda(u) = s$, where $s \in S$, by the disjunction of all formulas $u \in U_t$ such that $t \in T$ ends in s. This concludes the construction of the CDAA $\mathcal{B}$. The correctness proof can be found in the appendix.

5 Monadic Second-Order Logic

MSO logic over MSCs is two-sorted, as it shall reason about processes and events. By $u, v, w, \dots$ and $U, V, W, \dots$, we denote first-order and second-order variables, which range over processes and sets of processes, respectively. Moreover, by $x, y, z, \dots$ and $X, Y, Z, \dots$, we denote variables ranging over (sets of, respectively) events. The logic MSO_m is given by the grammar $\varphi ::= u \overset{a\ b}{\longmapsto} v \mid u = v \mid u \in U \mid \exists u.\varphi \mid \exists U.\varphi \mid \neg\varphi \mid \varphi \vee \varphi \mid x \lhd_{\mathsf{proc}} y \mid x \lhd_{\mathsf{msg}} y \mid x = y \mid x@u \mid x \in X \mid \exists x.\varphi \mid \exists X.\varphi$ where $a, b \in \mathcal{N}$.

MSO_m formulas are interpreted over MSCs $M = (P, \longmapsto, E, \lhd, \pi)$. Hereby, free variables u and x are interpreted by a function $\mathcal{I}$ as a process $\mathcal{I}(u) \in P$ and an event $\mathcal{I}(x) \in E$, respectively. Similarly, U and X are interpreted as sets. We write $M, \mathcal{I} \models u \overset{a\ b}{\longmapsto} v$ if $\mathcal{I}(u) \overset{a\ b}{\longmapsto} \mathcal{I}(v)$ and $M, \mathcal{I} \models x@u$ if $\pi(\mathcal{I}(x)) = \mathcal{I}(u)$. Thus, $x@u$ says that "x is located at u". The semantics of other formulas is as expected. When φ is a sentence, i.e., a formula without free variables, then its truth value is independent of an interpretation function so that we can simply write $M \models \varphi$ instead of $M, \mathcal{I} \models \varphi$. The set of MSCs M such that $M \models \varphi$ is denoted by $L(\varphi)$.

▶ **Example 15.** Let us resume the token-ring protocol from Example 6. We would like to express that there is a process that emits a message and gets an acknowledgment that results from a sequence of forwards through interface a. We first let $\mathrm{fwd}(x, y) \equiv x \overset{a\ b}{\longmapsto} y \wedge \exists z.(x \lhd_{\mathsf{proc}} z \lhd_{\mathsf{msg}} y)$ where $x \overset{a\ b}{\longmapsto} y$ is a shorthand for $\exists u.\exists v.(x@u \wedge y@v \wedge u \overset{a\ b}{\longmapsto} v)$. It is well known that the transitive closure of the relation induced by $\mathrm{fwd}(x, y)$ is definable in MSO_m-logic, too. Let $\mathrm{fwd}^+(x, y)$ be the corresponding formula. It expresses that there is a sequence of events leading from x to y that alternatingly takes process and message edges, hereby following the causal order. With this, the desired formula is $\varphi \equiv \exists x, y, z.(x \lhd_{\mathsf{proc}} y \wedge x \lhd_{\mathsf{msg}} z \wedge x \overset{a\ b}{\longmapsto} z \wedge \mathrm{fwd}^+(z, y)) \in \mathrm{MSO}_m$. Consider Figures 3 and 4. We have $M_1 \models \varphi$ and $M_2 \not\models \varphi$, as well as $L(\mathcal{A}'_{\mathrm{token}}) \subseteq L(\varphi)$. ◀

▶ **Theorem 16.** *Let $ct \in \{\mathsf{s{\oplus}r1}, \mathsf{intf}\}$, $k \geq 1$, and $L \subseteq \mathbb{MSC}$. There is a PCA $\mathcal{A}$ such that $L(\mathcal{A}) \equiv_{(k,ct)} L$ iff there is a sentence $\varphi \in \mathrm{MSO}_m$ such that $L(\varphi) \equiv_{(k,ct)} L$.*

The direction "$\Longrightarrow$" follows a standard pattern and is actually independent of a context bound. For the direction "$\Longleftarrow$", we proceed by induction, crucially relying on Theorem 9. Note that there are some subtleties in the translation, which arise from the fact that MSO_m mixes event and process variables (cf. appendix).

By the results from [4] and the fact that PCAs are closed under intersection (cf. [3]), we obtain decidability of MSO model checking as a corollary.

▶ **Theorem 17.** *Let $\mathfrak{T}$ be one of the following: the class of rings, the class of pipelines, or the class of ranked trees. The following problem is decidable, for all $ct \in \{\mathsf{s{\oplus}r1}, \mathsf{intf}\}$:*

Input: *A PCA* $\mathcal{A}$*, a sentence* $\varphi \in \mathrm{MSO}_m$*, and* $k \geq 1$.
Question: *Do we have* $M \models \varphi$ *for all MSCs* $M = (P, \longmapsto, E, \lhd, \pi) \in L(\mathcal{A}) \cap \mathbb{MSC}_{(k,ct)}$ *such that* $(P, \longmapsto) \in \mathfrak{T}$?

6 Conclusion

This paper constitutes a further step towards a language theory of parameterized concurrent systems. We established that PCAs are closed under complementation when processes are constrained to execute a bounded number of (suitable) contexts. As a consequence, we obtain that context-bounded PCAs are expressively equivalent to MSO logic.

Note that MSO logic is a powerful logic and it may actually be used for the verification of extended models that may, for example, involve registers to store process identities that can be checked for equality. MSO logic allows one to trace back the origin of a process identity so that an additional equality predicate on process identities can be reduced to an MSO formula over a finite alphabet. This would allow us to model and verify leader-election protocols. It will be worthwhile to explore this in future work.

References

1 P. A. Abdulla, F. Haziza, and L. Holík. All for the price of few. In *VMCAI'13*, volume 7737 of *LNCS*, pages 476–495. Springer, 2013.
2 B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI'14*, volume 8318 of *LNCS*, pages 262–281, 2014.
3 B. Bollig. Logic for communicating automata with parameterized topology. In *CSL-LICS'14*. ACM, 2014. to appear.
4 B. Bollig, P. Gastin, and J. Schubert. Parameterized Verification of Communicating Automata under Context Bounds. In *RP'14*, LNCS. Springer, 2014. to appear.
5 D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2), 1983.
6 G. Delzanno, A. Sangnier, and G. Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *FoSSaCS'11*, volume 6604 of *LNCS*, pages 441–455. Springer, 2011.
7 E. A. Emerson and K. S. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003.
8 J. Esparza. Keeping a crowd safe: On the complexity of parameterized verification. In *STACS'14*, volume 25 of *LIPIcs*, pages 1–10, 2014.
9 J. Esparza, P. Ganty, and R. Majumdar. Parameterized verification of asynchronous shared-memory systems. In *CAV'13*, LNCS, pages 124–140, 2013.
10 B. Genest, D. Kuske, and A. Muscholl. On communicating automata with bounded channels. *Fundam. Inform.*, 80(1-3):147–167, 2007.
11 S. La Torre, P. Madhusudan, and G. Parlato. The language theory of bounded context-switching. In *LATIN'10*, volume 6034 of *LNCS*, pages 96–107. Springer, 2010.
12 S. La Torre, M. Napoli, and G. Parlato. Scope-bounded pushdown languages. In *Proceedings of DLT'14*, LNCS. Springer, 2014. to appear.
13 N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.
14 O. Matz, N. Schweikardt, and W. Thomas. The monadic quantifier alternation hierarchy over grids and graphs. *Information and Computation*, 179(2):356–383, 2002.
15 S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS'05*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.
16 W. Thomas. Elements of an automata theory over partial orders. In *POMIV'96*, volume 29 of *DIMACS*. AMS, 1996.

A Proof of Theorem 8

First, note that the non-complementability result from [3] does not apply to our case of rendez-vous communication.

Our proof uses the fact that grid automata (or, *graph acceptors*) over grids are not complementable [14, 16]. Grids are rectangular structures that are uniquely determined by their height $m \geq 1$ and their width $n \geq 1$. Formally, a grid is a structure of the form $G = (C, \rightarrow, \downarrow)$ where $C = [m] \times [n]$ is the set of *coordinates*, $\rightarrow = \{((i,j),(i,j+1)) \mid i \in [m] \text{ and } j \in [n-1]\}$ is the "go to the right"-relation, and $\downarrow = \{((i,j),(i+1,j)) \mid i \in [m-1] \text{ and } j \in [n]\}$ is the "go down"-relation. The type $\mathit{type}(i,j)$ of a coordinate $(i,j) \in C$ is a subset of the collection $D = \{\mathsf{n}, \mathsf{s}, \mathsf{w}, \mathsf{e}\}$ of *directions* (denoting north, south, west, and east, respectively). We let $\mathsf{n} \in \mathit{type}(i,j)$ iff $i \geq 2$ and $\mathsf{w} \in \mathit{type}(i,j)$ iff $j \geq 2$. Moreover, $\mathsf{s} \in \mathit{type}(i,j)$ iff $i < m$ and $\mathsf{e} \in \mathit{type}(i,j)$ iff $j < n$. We let $d(i,j)$ be the d-neighbor of (i,j), which is defined in the obvious manner for all $d \in \mathit{type}(i,j)$. By $G_{m,n}$, we denote the grid with set of coordinates $[m] \times [n]$. Finally, let $\mathbb{G}$ be the set of all grids.

A *grid automaton* is a pair $\mathcal{G} = (S, \Delta)$ where S is the nonempty finite set of states and Δ is the set of transitions. A transition labels a coordinate with a state, determines its type, and assigns a state to each of its neighbors. It is given by a pair (s, ν) where $s \in S$ and $\nu : D \rightharpoonup S$ is a partial mapping. A run of $\mathcal{G}$ on grid $G = (C, \rightarrow, \downarrow)$ is a labeling $\rho : C \to S$ of coordinates with states that is consistent with Δ: for all $(i,j) \in C$, there is a transition $(s, \nu) \in \Delta$ such that $s = \rho(i,j)$, $\mathrm{dom}(\nu) = \mathit{type}(i,j)$, and $\nu(d(i,j)) = \rho(d(i,j))$ for all $d \in \mathit{type}(i,j)$. Note that there is no further acceptance condition, i.e., every run is accepting. By $L(\mathcal{G})$, we denote the set of grids that allow for a run of $\mathcal{G}$. Actually, grid automata are expressively equivalent to graph acceptors (running on grids), which have an acceptance condition and transitions with a larger action radius [16].

▶ **Theorem 18** (Matz et al. [14]; Thomas [16]). *There is a grid automaton $\mathcal{G}$ such that no $\mathcal{G}'$ exists with $L(\mathcal{G}') = \mathbb{G} \setminus L(\mathcal{G})$.*

Grids can be encoded into MSCs as depicted in Figures 7 and 8 for context types $\mathsf{s1{+}r1}$ and $\mathsf{s}\oplus\mathsf{r}$, respectively. Note that both MSCs use only one context per process. Actually, there are several ways to interpret these MSCs as grids. In both cases, we may say that events of type $a!$ correspond to the coordinates and that, roughly, going down on a process line corresponds to going down in the grid. Thus, the MSC from Figure 7 encodes $G_{4,4}$, and the MSC from Figure 8 encodes $G_{4,2}$.

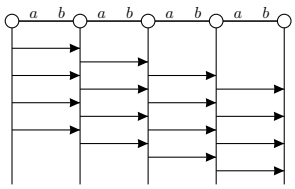


Figure 7 Grid encoding for $\mathsf{s1{+}r1}$

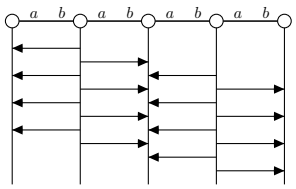


Figure 8 Grid encoding for $\mathsf{s}\oplus\mathsf{r}$

We assume context type $\mathsf{s1{+}r1}$, the case of $\mathsf{s}\oplus\mathsf{r}$ is similar. For $G \in \mathbb{G}$, let $\mathit{msc}(G)$ denote the MSC encoding of G. Conversely, if $M = \mathit{msc}(G)$ then we let $\mathit{grid}(M) = G$ (which is well-defined). Both mappings extend to sets as expected. Note that we can easily construct a PCA $\mathcal{A}_{\mathrm{grid}}$ over the interface set $\{a, b\}$ such that $L(\mathcal{A}_{\mathrm{grid}}) = \mathit{msc}(\mathbb{G})$, i.e., $\mathcal{A}$ accepts precisely the MSCs that encode a grid.

The following lemmas state that PCAs can simulate grid automata, and vice versa.

▶ **Lemma 19.** *For every grid automaton $\mathcal{G}$, there is a PCA $\mathcal{A}$ such that $L(\mathcal{A}) = msc(L(\mathcal{G}))$.*

Proof (sketch). Initially, the PCA $\mathcal{A}$ guesses whether its process is the leftmost one, an inner process or the rightmost one. It keeps this guess in its states and the correctness of the guess is checked by the acceptance condition of $\mathcal{A}$. Below, we describe the behavior of the PCA at an inner process. The cases of a left-most or right-most process are similar.

At every send event, the automaton guesses the transition (s, ν) that is used by the grid automaton $\mathcal{G}$ at this coordinate. In order to check that the guesses form a run of $\mathcal{G}$, some information is propagated along process edges and message edges. A send event e with guessed transition (s, ν) propagates the pair $(s, \nu(\mathsf{e}))$ to its matching receive event and (if it is not the last event on its process) the pair $(s, \nu(\mathsf{s}))$ to its process successor which is also a read event.

Hence a read event f receives a pair $(s_\mathsf{w}, t_\mathsf{w})$ as a message from its matching send event and a pair $(s_\mathsf{n}, t_\mathsf{n})$ from its process predecessor (if f is not the first event on its process). It checks that $t_\mathsf{w} = t_\mathsf{n} = t$ and propagates the triple $(s_\mathsf{w}, s_\mathsf{n}, t)$ to its process successor which is a send event e'. The automaton $\mathcal{A}$ then makes sure that the transition (s', ν') of $\mathcal{G}$ guessed by e' is consistent with $s' = t$, $\nu'(\mathsf{n}) = s_\mathsf{n}$ and $\nu'(\mathsf{w}) = s_\mathsf{w}$.

If a read event f is the first on its process it propagates the pair $(s_\mathsf{w}, t_\mathsf{w})$ to its process successor e'. In such a case, the transition (s', ν') guessed by e' should satisfy $s' = t_\mathsf{w}$, $\nu'(\mathsf{w}) = s_\mathsf{w}$ and $\mathsf{n} \notin \mathrm{dom}(\nu')$.

Also, the last send event on its process guesses a transition (s, ν) with $\mathsf{s} \notin \mathrm{dom}(\nu)$ and the correctness of this guess is checked by the acceptance condition of $\mathcal{A}$. ◀

▶ **Lemma 20.** *For every PCA $\mathcal{A}$ such that $L(\mathcal{A}) \subseteq msc(\mathbb{G})$, there is a grid automaton $\mathcal{G}$ such that $L(\mathcal{G}) = grid(L(\mathcal{A}))$.*

Proof (sketch). There are two main issues in the translation of the PCA $\mathcal{A} = (S, \iota, Msg, \Delta, \mathcal{F})$ into the grid automaton $\mathcal{G}$.

First, $\mathcal{G}$ has to simulate the transitions of $\mathcal{A}$ applied to an MSC $M \in msc(\mathbb{G})$ using only the nodes of the encoded grid G. Recall that every node in G corresponds to an event in M, while the other direction fails. However, similarly to the proof of Lemma 19, we can encode the transitions of $\mathcal{A}$. The set of states of $\mathcal{G}$ is $S \times Msg$. A transition $((s, m), \nu)$ of $\mathcal{G}$ with, for example, $\{\mathsf{n}, \mathsf{w}\} \subseteq \mathrm{dom}(\nu)$ checks if $\mathcal{A}$ has transitions of the form $(s_b, (b?, m_a), s'_b)$ and $(s'_b, (a!, m), s)$ such that $\nu(\mathsf{w}) = (s_a, m_a)$ for some s_a, and $\nu(\mathsf{n}) = (s_b, m')$ for some m'. Adding messages to states will make sure that, at the w-neighbor of the current event, m_a is indeed the emitted message, so that a run of $\mathcal{G}$ gives rise to a consistent run of $\mathcal{A}$.

Second, the acceptance condition of $\mathcal{A}$ has to be taken care of by $\mathcal{B}$, which itself does not have an acceptance condition. Recall that we have to consider only MSCs M that are encodings of grids. The underlying topology is a pipeline, and every process executes at least one event. By the Büchi-Elgot-Trakhtenbrot theorem, the acceptance condition $\mathcal{F}$ of $\mathcal{A}$ thus reduces to a regular language over S. In turn, the grid automaton can simulate the corresponding finite automaton over S on the last row of the grid. ◀

According to Theorem 18, let $\mathcal{G}$ be a grid automaton such that no $\mathcal{G}'$ exists with $L(\mathcal{G}') = \mathbb{G} \setminus L(\mathcal{G})$. By Lemma 19, there is a PCA $\mathcal{A}$ such that $L(\mathcal{A}) = msc(L(\mathcal{G}))$. Towards a contradiction, suppose that PCAs are $\mathsf{s1}$+$\mathsf{r1}$-complementable. Then, there is a PCA $\mathcal{A}''$ such that $L(\mathcal{A}'') \equiv_{(1,\mathsf{s1}+\mathsf{r1})} \mathbb{MSC} \setminus msc(L(\mathcal{G}))$. Building the product with $\mathcal{A}_{\text{grid}}$, we transform $\mathcal{A}''$ into a PCA $\mathcal{A}'$ satisfying $L(\mathcal{A}') = msc(\mathbb{G}) \setminus msc(L(\mathcal{G}))$. By Lemma 20, there is a grid

automaton $\mathcal{G}'$ such that $L(\mathcal{G}') = \mathit{grid}(\mathit{msc}(\mathbb{G}) \setminus \mathit{msc}(L(\mathcal{G})))$. The latter equals $\mathbb{G} \setminus L(\mathcal{G})$, which is a contradiction to the assumption that $\mathcal{G}$ is not complementable.

B Missing Details for Proof of Lemma 13

We start by showing that $\mathcal{B}$ computes zones when it is run on a $(k, \mathsf{s}{\oplus}\mathsf{r1})$-bounded MSC.

▶ **Lemma 21.** *Let $M = (P, \longmapsto, E, \lhd, \pi) \in \mathbb{MSC}_{(k,\mathsf{s}\oplus\mathsf{r1})}$ and let ρ' be the unique run of $\mathcal{B}$ on M. Then, ρ' is a mapping $E \to S'$ (i.e., it does not use any sink state), and the equivalence relation $\sim$ defined as follows induces a zone partitioning of E.*

For $e \in E$, we let in the following $(i_e, \tau_e, \kappa_e, R_e) \in Z$ denote the last zone state in the sequence $\rho'(e)$. We define $\sim \subseteq E \times E$ as the least equivalence relation such that $\lhd_{\mathsf{msg}} \subseteq \sim$ and $\{(e, e') \in \lhd_{\mathsf{proc}} \mid i_e = i_{e'}\} \subseteq \sim$.

Proof. Pick $p \in P$. We partition the set of *send* events on p into *intervals* as follows. The first interval I contains the first send event (wrt. $\lhd^*_{\mathsf{proc}}$). We add further send events to I as long as, for all $a \in \mathcal{N}$, the corresponding receive events at the a-neighbor also form an interval I_a. Then, consider the next maximal interval, and so on. When we are done with p, we pick another process, etc. We will show that the zone partitioning induced by this procedure coincides with the equivalence classes of $\sim$.

Let $p \in P$. Let I be one of the constructed send intervals on p and suppose that its receive intervals are given by I_a, for each $a \in \mathcal{N}$ (note that I_a may be empty). By definition, $\mathcal{Z} = I \cup \bigcup_{a \in \mathcal{N}} I_a$ is a zone. It is also easily seen that, by (1) and (2) in the definition of the functions $\delta^{\mathrm{zone}}_{(a,b)}$, we have $e \sim e'$ for every two events $e, e' \in \mathcal{Z}$. It remains to show that $\mathcal{Z}$ defines a *whole* equivalence class of $\sim$, i.e., whenever $e \in \mathcal{Z}$ and $e' \notin \mathcal{Z}$, then $e \not\sim e'$.

We define $\min(I), \max(I) \in E$ in the obvious way. Suppose $e \in E$ such that $e \lhd_{\mathsf{proc}} \min(I)$. Assume that I is the first send interval on p. By (1), we have that $i_e < i_{\min(I)}$. Now, assume that I is not the first interval that we chose on p. If e is a receive event, then $i_e < i_{\min(I)}$ by (1). So, suppose that e is a send event. Then, e belongs to the previous send interval on p that we have chosen. Assume $e \lhd_{\mathsf{msg}} f$ and $p = \pi(e) \overset{a\ b}{\longmapsto} \pi(f)$. Suppose $e' = \min(I)$ also sends through a, say, with $e' \lhd_{\mathsf{msg}} f'$. Then, f and f' do not belong to the same context. By (1) and (2), we have $i_f < i_{f'}$, which implies $i_e < i_{e'}$. Now, suppose e' sends through $b \neq a$ and that f' is the matching receive event. Then, there is a send event $\hat{e}$ from the send interval preceding I (which also contains e) such that $\ell_M(\hat{e}) = b!$ and whose corresponding receive event $\hat{f}$ is not in the same context as f'. Suppose $(\hat{e}, \hat{f})$ is the last pair before I with that property. By (1) and (2), we have $i_{\hat{f}} = \kappa_e(b) < i_{f'}$. Again, this implies $i_e < i_{e'}$.

Suppose $e \in E$ such that $\max(I) \lhd_{\mathsf{proc}} e$. If e is a receive event, then $i_{\max(I)} < i_e$ by (2). Now assume that e is a send event and f is such that $e \lhd_{\mathsf{msg}} f$ and $\pi(e) \overset{a\ b}{\longmapsto} \pi(f)$. Then, by (2) and the maximality of I, we have $I_a \neq \emptyset$ and $i_{f'} < i_f$ for all $f' \in I_a$. By (1), this implies $i_{\max(I)} < i_e$.

Next, pick one of the intervals I_a associated with I. By a similar reasoning, we get that, for all $e \in E$, $e \lhd_{\mathsf{proc}} \min(I_a)$ implies $i_e < i_{\min(I_a)}$, and $\max(I_a) \lhd_{\mathsf{proc}} e$ implies $i_{\max(I_a)} < i_e$. Recall that, for all $e, f \in E$ with $\pi(e) \overset{a\ b}{\longmapsto} \pi(f)$ and $e \lhd_{\mathsf{msg}} f$, we have $\kappa_e(a) = i_f$ and $\kappa_f(b) = i_e$. Thus, we have shown that $\mathcal{Z}$ coincides with an equivalence class of $\sim$.

When we partition the set of send events on a process p, we may have to divide some send contexts, since the corresponding receive events on some a-neighbor do not necessarily belong to one context. However, for each neighbor, we have at most k such splittings. In turn, $|\mathcal{N}| \cdot (k + k \cdot |\mathcal{N}|)$ new receive contexts may be created on p by splitting send contexts in neighboring processes. As a result, process p traverses at most $K := k \cdot (|\mathcal{N}|^2 + 2|\mathcal{N}| + 1)$ different zones (cf. [4]). We deduce that $\rho'(e) \in S'$ for all $e \in E$. ◀

We are now ready to show that $L(\mathcal{A}) \equiv_{(k,\mathsf{s}\oplus\mathsf{r}1)} L(\mathcal{B})$. So let us fix an MSC $M = (P, \longmapsto, E, \lhd, \pi) \in \mathbb{MSC}_{(k,\mathsf{s}\oplus\mathsf{r}1)}$. When, in the following, we talk about a zone of M, we mean a zone induced by $\sim$ as defined in Lemma 21. We start with the reverse inclusion which is the most challenging.

▶ **Lemma 22.** *If $M \in L(\mathcal{B})$ then $M \in L(\mathcal{A})$.*

Proof. Since $\mathcal{B}$ is deterministic and complete, there is a unique run of $\mathcal{B}$ on M, which is given by $\rho' : E \to S'$ (by Lemma 21, since M is $(k, \mathsf{s}\oplus\mathsf{r}1)$-bounded, the run does not use any sink state). Let $\lambda' : P \to S'$ denote the local final state at each process for ρ'. We know that $(P, \longmapsto, \lambda')$ satisfies $\mathcal{F}'$. Recall that $\mathcal{F}'$ guesses an assignment $\sigma : P \to T$ by means of the second-order variables U_t for $t \in T$. We have to construct an accepting run $\rho : E \to S$ of $\mathcal{A}$ for M.

For each process p, let n_p be the number of zones of process p computed by $\mathcal{B}$. Then, we have $\sigma(p) = \iota\sigma(p)_1 \cdots \sigma(p)_{n_p}$. It is easy to assign a state from S to the last event of each zone: for all $i \in [n_p]$, if $e_{p,i}$ is the last event of the i-th zone of process p then we let $\rho(e_{p,i}) = \sigma(p)_i$. Then, we have to assign states from S to the intermediary events of each zone. To this end, we will use the following lemma.

▶ **Lemma 23.** *Let $\lambda'(p) = (0, \emptyset, \kappa_0, \emptyset)(1, \tau_1, \kappa_1, R_1) \dots (n_p, \tau_{n_p}, \kappa_{n_p}, R_{n_p})$. Let $i \in [n_p]$ be such that $\tau_i \subseteq \mathcal{N}$ (in its i-th zone, process p is sending). Then, for all $(\bar{s}, \bar{s}') \in R_i$ there is a run of $\mathcal{A}$ on the i-th zone of process p starting from $\bar{s}$ and ending in $\bar{s}'$.*

Proof. We construct the run by induction on the number of send messages added in this i-th zone. The zone is created with a first message sent on some interface $p \overset{a\ b}{\longmapsto} q$. The update function $\delta^{\text{zone}}_{(a,b)}$ starts a new zone with the relation $R'_1 = R$ defined in (3). By definition, for each $(\bar{s}, \bar{s}') \in R$, there is a message $m \in Msg$ with $(\bar{s}_{\mathsf{self}}, (a!, m), \bar{s}'_{\mathsf{self}}) \in \Delta$, $(\bar{s}_a, (b?, m), \bar{s}'_a) \in \Delta$, and $\bar{s}_c = \bar{s}'_c$ for all $c \in \mathcal{N} \setminus \{a\}$. Hence, $\mathcal{A}$ has a run from $\bar{s}$ to $\bar{s}'$ on the zone consisting of this single message sent from p to q.

Assume now that the nonempty i-th zone is extended via $\delta^{\text{zone}}_{(a,b)}$ with a new message sent on $p \overset{a\ b}{\longmapsto} q$. Then, $R'_1 = R_1 \circ R$ according to (3). Let $(\bar{s}, \bar{s}'') \in R'_1$. We have $(\bar{s}, \bar{s}') \in R_1$ and $(\bar{s}', \bar{s}'') \in R$ for some state $\bar{s}' \in S^{\mathcal{N} \cup \{\mathsf{self}\}}$. By induction, we have a run of $\mathcal{A}$ from $\bar{s}$ to $\bar{s}'$ on the zone before adding the new message. By definition of R, we also have a run of $\mathcal{A}$ from $\bar{s}'$ to $\bar{s}''$ on the last message sent from p to q. This results in a run of $\mathcal{A}$ from $\bar{s}$ to $\bar{s}''$ on the zone exented with the new message from p to q. ◀

We continue the proof of Lemma 22. We have already defined the states assigned to the last event of each zone $\rho(e_{p,i}) = \sigma(p)_i$. We use the notation of Lemma 23 with $i \in [n_p]$ being a send-zone for process p. Since the run of $\mathcal{B}$ is accepting, $(P, \longmapsto, \lambda') \models \mathcal{F}'$ and we have chosen the assignment σ which witnesses this fact. Therefore, there is $(\bar{s}, \bar{s}') \in R_i$ such that (i) $\bar{s}_{\mathsf{self}} = \sigma(p)_{i-1}$ and $\bar{s}'_{\mathsf{self}} = \sigma(p)_i$, and (ii) for all $p \overset{a\ b}{\longmapsto} q$ we have $\bar{s}_a = \sigma(q)_{\kappa_i(a)-1}$ and $\bar{s}'_a = \sigma(q)_{\kappa_i(a)}$ if $a \in \tau_i$, and $\bar{s}'_a = \bar{s}_a = \sigma(q)_{\kappa_i(a)}$ if $a \notin \tau_i$. We apply Lemma 23 to the pair $(\bar{s}, \bar{s}')$ and we obtain a run of $\mathcal{A}$ from $\bar{s}$ to $\bar{s}'$ on the i-th zone of process p. We use this run to extend the map ρ to all events of the i-th zone of process p. We repeat this construction for all send-zones and we obtain a fully defined map $\rho : E \to S$. By construction, the restriction of ρ to each send-zone defines a run of $\mathcal{A}$, hence ρ is a run of $\mathcal{A}$ on M.

It remains to show that ρ is accepting. By construction, the last state of the run ρ for process p is $\lambda(p) = \sigma(p)_{n_p}$ which is the last state of $\sigma(p)$ (with the convention $\sigma(p)_0 = \iota$). Now, the acceptance condition $\mathcal{F}'$ contains (as a conjunction) a formula $\mathcal{F}''$ which consists of the acceptance condition $\mathcal{F}$ in which each atomic formula $\lambda(u) = s$ with $s \in S$ is replaced with the disjunction of all formulas $u \in U_t$ such that $t \in T$ ends in s. Let $\mathcal{I}'$ be the interpretation

of the variables $(U_t)_{t\in T}$ defined by σ. Notice that $p \in \mathcal{I}'(U_{\sigma(p)})$ for all $p \in P$ and that $\sigma(p)$ ends with $\lambda(p)$. Assume that a first-order process variable u is assigned to process p under some interpretation. Then, $\lambda(u) = s$ holds iff $s = \lambda(p) = \sigma(p)_{n_p}$ iff $u \in U_t$ holds for some t that ends with s (since the only such t is $\sigma(p)$). Then, it is not difficult to check by induction on the formula that $(P, \longmapsto, \lambda'), \mathcal{I}' \models \mathcal{F}''$ if and only if $(P, \longmapsto, \lambda) \models \mathcal{F}$. ◀

▶ **Lemma 24.** *If $M \in L(\mathcal{A})$ then $M \in L(\mathcal{B})$.*

Proof. Assume that $M \in L(\mathcal{A})$ and let $\rho : E \to S$ be an accepting run of $\mathcal{A}$ on M. Let $\lambda : P \to S$ denote the local final state of ρ at each process. Then, $(P, \longmapsto, \lambda)$ satisfies $\mathcal{F}$.

Since $\mathcal{B}$ is deterministic and complete, there is a unique run of $\mathcal{B}$ on M. By Lemma 21, since M is $(k, \mathsf{s}{\oplus}\mathsf{r1})$-bounded, the run does not end in a sink state so that it is a mapping $\rho' : E \to S'$. Let $\lambda' : P \to S'$ denote the local final state at each process for ρ'.

For $p \in P$, assume that $\lambda'(p) = (0, \emptyset, \kappa_0, \emptyset)(1, \tau_1, \kappa_1, R_1) \dots (n_p, \tau_{n_p}, \kappa_{n_p}, R_{n_p})$. The events of process p have been partitioned into n_p zones. Let $\sigma(p)_0 = \iota$ and, for $i \in [n_p]$, $\sigma(p)_i = \rho(e_i)$ where e_i is the last event of the i-th zone on process p. Then, we let $\sigma(p) = \sigma(p)_0\sigma(p)_1 \cdots \sigma(p)_{n_p}$. The assignment $\sigma : P \to T$ is well-defined and induces an interpretation of each second-order variable U_t for $t \in T$ as the set $\sigma^{-1}(t) \subseteq P$.

Pick $p \in P$ and suppose $\lambda'(p) = (0, \emptyset, \kappa_0, \emptyset)(1, \tau_1, \kappa_1, R_1) \dots (n_p, \tau_{n_p}, \kappa_{n_p}, R_{n_p})$. Furthermore, let $i \in [n_p]$ with $\tau_i \subseteq \mathcal{N}$. We define $\bar{s}, \bar{s}' \in S^{\mathcal{N} \cup \{\mathsf{self}\}}$ by (i) $\bar{s}_{\mathsf{self}} = \sigma(p)_{i-1}$ and $\bar{s}'_{\mathsf{self}} = \sigma(p)_i$, (ii) $\bar{s}_a = \sigma(q)_{\kappa_i(a)-1}$ and $\bar{s}'_a = \sigma(q)_{\kappa_i(a)}$ if $a \in \tau_i$, and $\bar{s}_a = \bar{s}'_a = \sigma(q)_{\kappa_i(a)}$ if $a \notin \tau_i$. Note that the i-th zone of p involves those processes q such that $p \overset{a\ b}{\longmapsto} q$ with $a \in \tau_i$ and $b \in \mathcal{N}$. Since ρ is a run of $\mathcal{A}$ on M, there is a run of $\mathcal{A}$ on the i-th zone of p that starts from $\bar{s}$ and ends in $\bar{s}'$. By Equation (3), since processes do not change their zone number within a zone, we have $(\bar{s}, \bar{s}') \in R_i$.

Now, consider $(P, \longmapsto, \lambda)$ with the interpretation induced by σ. Since $\sigma(p) \in T$ keeps track of the states visited at the end of each zone and $(P, \longmapsto, \lambda)$ satisfies $\mathcal{F}$, we have that $(P, \longmapsto, \lambda')$ also satisfies $\mathcal{F}$ where each atomic subformula $\lambda(u) = s$ ($s \in S$) is replaced by the disjunction of all formulas $u \in X_t$ such that $t \in T$ ends in s. ◀

C The Case of Context Type intf

When $ct = \mathsf{intf}$, we define the zones in a slightly different way.

Zones. Let $M = (P, \longmapsto, E, \lhd, \pi)$ be an MSC. An intf-*zone* is a nonempty set of events of the form $\{e_1, \dots, e_n, f_1, \dots, f_n\} \subseteq E$ such that

- $e_1 \lhd_{\mathsf{proc}} e_2 \lhd_{\mathsf{proc}} \dots \lhd_{\mathsf{proc}} e_n$,
- $f_1 \lhd_{\mathsf{proc}} f_2 \lhd_{\mathsf{proc}} \dots \lhd_{\mathsf{proc}} f_n$, and
- for all $i \in [n]$, either $e_i \lhd_{\mathsf{msg}} f_i$ or $f_i \lhd_{\mathsf{msg}} e_i$.

▶ **Lemma 25.** *[cf. [4]] Let $M = (P, \longmapsto, E, \lhd, \pi)$ be a (k, intf)-bounded MSC. There is a partitioning of the events of M into zones such that, for each process $p \in P$, the events of E_p belong to at most $K := k + (k \cdot |\mathcal{N}|)$ zones.*

Here also, a process stores the zone number of each of its neighbors. A *process enters a new zone* if (a) it starts receiving from or sending to a different process, or (b) the zone number of its neighbor does not match. A zone state from Z is a tuple (i, τ, κ, R) where i has the same meaning as before, but $\tau \in \mathcal{N}$, $\kappa \in \{0, \dots, K\}$, and $R \subseteq (S \times S)^2$. The component τ carries information only about the current neighbor and κ stores its zone number. Moreover, R is the set of possible global steps that the zone may induce.

We now define the partial function $\delta^{\text{zone}}_{(a,b)} : (Z \times Z) \rightharpoonup (Z \times Z)$ for $(a,b) \in \mathcal{N} \times \mathcal{N}$ by

$$\delta^{\text{zone}}_{(a,b)}((i_1,\tau_1,\kappa_1,R_1),(i_2,\tau_2,\kappa_2,R_2)) = ((i_1',a,i_2',R_1'),(i_2',b,i_1',R_2'))$$

where

$$i_1' = \begin{cases} i_1+1 & \text{if } i_1 = 0 \text{ or } \tau_1 \neq a \text{ or } (\tau_1 = a \text{ and } \kappa_1 \neq i_2) \\ i_1 & \text{otherwise} \end{cases}$$

$$i_2' = \begin{cases} i_2+1 & \text{if } i_2 = 0 \text{ or } \tau_2 \neq b \text{ or } (\tau_2 = b \text{ and } \kappa_2 \neq i_1) \\ i_2 & \text{otherwise} \end{cases}$$

$$R_1' = \begin{cases} R & \text{if } i_1' = i_1+1 \\ R_1 \circ R & \text{otherwise} \end{cases} \qquad R_2' = \begin{cases} R' & \text{if } i_2' = i_2+1 \\ R_2 \circ R' & \text{otherwise} \end{cases}$$

with R being the set of pairs $((s_a,s_b),(s_a',s_b')) \in (S \times S)^2$ such that there is $m \in Msg$ with $(s_a,(a!,m),s_a') \in \Delta$ and $(s_b,(b?,m),s_b') \in \Delta$. Also, R' is the set of pairs $((s_b,s_a),(s_b',s_a')) \in (S \times S)^2$ such that $((s_a,s_b),(s_a',s_b')) \in R$. Note that the zone switch happens simultaneously for both the neighbors.

We will now define the transition function $\delta_{(a,b)} : (S' \times S') \to (S' \times S')$ where S' is the set of words over Z of the form $(0,\mathsf{a},0,\emptyset)(1,\tau_1,\kappa_1,R_1)\dots(n,\tau_n,\kappa_n,R_n)$ where $n \in \{0,\dots,K\}$ and $\mathsf{a} \in \mathcal{N}$ is an arbitrary fixed interface. The initial state is $\iota' = (0,\mathsf{a},0,\emptyset)$. Let $z_1 = (i_1,\tau_1,\kappa_1,R_1) \in Z$ and $z_2 = (i_2,\tau_2,\kappa_2,R_2) \in Z$. Also, suppose $\delta^{\text{zone}}_{(a,b)}(z_1,z_2) = (z_1',z_2')$ where $z_1' = (i_1',\tau_1',\kappa_1',R_1')$ and $z_2' = (i_2',\tau_2',\kappa_2',R_2')$. Then, we let

$$\delta_{(a,b)}(w_1z_1\,,\,w_2z_2) = \begin{cases} (w_1z_1'\,,\,w_2z_2') & \text{if } i_1' = i_1 \text{ and } i_2' = i_2 \\ (w_1z_1z_1'\,,\,w_2z_2z_2') & \text{if } i_1' = i_1+1 \text{ and } i_2' = i_2+1 \end{cases}$$

Again, the automaton is not complete, since $\delta_{(a,b)}$ is a partial function. However, we can again add a sink state to obtain a CDAA that is complete.

Acceptance Condition. Let T be the set of sequences of the form $\iota s_1 \dots s_n$ where $n \in \{0,\dots,K\}$ and $s_i \in S$ for all i. By means of second-order variables U_t, with t ranging over T, the formula $\mathcal{F}'$ guesses an assignment $\sigma : P \to T$. It will then check that, for all $p \in P$ with, say, $\lambda(p) = \iota'(1,\tau_1,\kappa_1,R_1)\dots(n,\tau_n,\kappa_n,R_n) \in S'$, the following hold: For all $i \in [n]$, there exists $((s_1,s_2),(s_1',s_2')) \in R_i$ such that

- $s_1 = \sigma(p)_{i-1}$ and $s_1' = \sigma(p)_i$, and
- there exist $b \in \mathcal{N}$ and $q \in P$ such that $p \xmapsto{\tau_i\ b} q$, $s_2 = \sigma(q)_{\kappa_i-1}$, and $s_2' = \sigma(q)_{\kappa_i}$.

The acceptance condition $\mathcal{F}$ is incorporated in the same way as in the previous case. The proof of correctness of the above construction is similar to that of Lemma 13.

D Proof of Theorem 16

We first consider the direction "$\Longrightarrow$". Let $\mathcal{A} = (S,\iota,Msg,\Delta,\mathcal{F})$ be a PCA. We construct a sentence $\varphi \in \text{MSO}_m$ such that $L(\mathcal{A}) = L(\varphi)$. Let $max(x)$ and $min(x)$ be formulas denoting that x is the first and, respectively, last event on its process. The formula φ will guess an assignment of events to states in terms of second-order variables $(X_s)_{s\in S}$. Similarly, to evaluate the acceptance condition $\mathcal{F}$ of the given PCA, we use second-order variables $(U_s)_{s\in S}$.

We let

$$\begin{aligned}
\varphi = \; & \exists (X_s)_{s\in S}.\exists (U_s)_{s\in S}. \\
& \quad \mathit{EventPart}((X_s)_{s\in S}) \wedge \mathit{ProcPart}((U_s)_{s\in S}) && (1)\\
& \wedge \quad \forall u.\forall x.(x@u \wedge \mathit{max}(x) \to \textstyle\bigwedge_{s\in S} u \in U_s \leftrightarrow x \in X_s) && (2)\\
& \wedge \quad \forall u.((\neg\exists x.x@u) \to u \in U_\iota) && (3)\\
& \wedge \quad \varphi_{\mathcal{F}} && (4)\\
& \wedge \quad \textstyle\bigwedge_{a,b\in\mathcal{N}} \forall x.\forall y.(x \overset{a\ b}{\longmapsto} y \wedge x \lhd_{\mathsf{msg}} y \to \bigvee_{m\in Msg} \mathit{trans}_m^{(a,b)}(x,y,(X_s)_{s\in S})) && (5)
\end{aligned}$$

The meaning of the subformulas is as follows:

(1) Formulas $\mathit{EventPart}((X_s)_{s\in S})$ and $\mathit{ProcPart}((U_s)_{s\in S})$ ensure that $(X_s)_{s\in S}$ and $(U_s)_{s\in S}$ form partitions of the set of events and, respectively, processes of the given MSC.

(2) The formula makes sure that, in U_s, we collect those processes whose maximal event terminates in state s.

(3) Similarly, U_ι should contain all processes that do not execute any event.

(4) Items (2) and (3) allow us to use a formula $\varphi_{\mathcal{F}}$ to simulate the acceptance condition $\mathcal{F}$ of $\mathcal{A}$. It is obtained from $\mathcal{F}$ by replacing every subformula of the form $\lambda(u) = s$ by $u \in U_s$.

(5) $x \overset{a\ b}{\longmapsto} y$ is a shorthand for $\exists u.\exists v.(x@u \wedge y@v \wedge u \overset{a\ b}{\longmapsto} v)$

(6) Finally, $\mathit{trans}_m^{(a,b)}(x,y,(X_s)_{s\in S}))$ makes sure that there are transitions that can be applied at x and y to exchange a message m through interfaces a and b. We have to handle several cases, depending on whether x and y are the first event of their process. This is formalized as follows:

$$\bigvee_{\substack{s_1,s_1',s_2,s_2'\in S \mid \\ (s_1,(a!,m),s_1')\in\Delta \\ (s_2,(b?,m),s_2')\in\Delta}} x \in X_{s_1'} \wedge y \in X_{s_2'} \wedge \left(\begin{array}{ll} & \exists \bar{x}.\exists \bar{y}.(\bar{x} \lhd_{\mathsf{proc}} x \wedge \bar{y} \lhd_{\mathsf{proc}} y \wedge \bar{x} \in X_{s_1} \wedge \bar{y} \in X_{s_2}) \\ \vee & \exists \bar{y}.(\mathit{min}(x) \wedge \bar{y} \lhd_{\mathsf{proc}} y \wedge \text{“}s_1 = \iota\text{”} \wedge \bar{y} \in X_{s_2}) \\ \vee & \exists \bar{x}.(\bar{x} \lhd_{\mathsf{proc}} x \wedge \mathit{min}(y) \wedge \bar{x} \in X_{s_1} \wedge \text{“}s_2 = \iota\text{”}) \\ \vee & \mathit{min}(x) \wedge \mathit{min}(y) \wedge \text{“}s_1 = \iota\text{”} \wedge \text{“}s_2 = \iota\text{”} \end{array}\right)$$

Here, formula “$s_i = \iota$” is either “true” or “false” depending on whether $s_i = \iota$ or not. This completes the construction of the formula $\varphi \in \mathrm{MSO}_m$ such that $L(\mathcal{A}) = L(\varphi)$.

Let us turn to direction “$\Longleftarrow$”. As we deal with an inductive translation, we have to cope with free variables and, thus, to consider an extension of MSCs that allows for an encoding of variables. As PCA are closed under projection, this can be safely done.

Consider a formula $\varphi \in \mathrm{MSO}_m$ with free variables $\mathsf{Free}(\varphi) \subseteq \mathcal{V} = \mathcal{U} \uplus \mathcal{X}$ where $\mathcal{U}$ consists of first-order and second-order *process* variables and $\mathcal{X}$ consists of first-order and second-order *event* variables.

$\mathcal{V}$-Extended MSCs are structures of the form (M, f, g) where $M = (P, \longmapsto, E, \lhd, \pi)$ is an MSC, $f : P \to \{0,1\}^{\mathcal{U}}$ and $g : E \to \{0,1\}^{\mathcal{X}}$. The pair (f, g) encodes an interpretation $\mathcal{I}$ of the variables in $\mathcal{V}$ in the following way. For second-order variables $U \in \mathcal{U}$ and $X \in \mathcal{X}$ we let $\mathcal{I}(U) = \{p \in P \mid f(p)(U) = 1\}$ and $\mathcal{I}(X) = \{e \in E \mid g(e)(X) = 1\}$. For first-order variables $u \in \mathcal{U}$ and $x \in \mathcal{X}$ we require that there is exactly one process $p \in P$ (resp. event $e \in E$) such that $f(p)(u) = 1$ (resp. $g(e)(x) = 1$) and we let $\mathcal{I}(u) = p$ (resp. $\mathcal{I}(x) = e$).

To handle such an interpretation $\mathcal{I}$ of free variables, we consider $\mathcal{V}$-PCAs running on $\mathcal{V}$-extended MSCs (M, f, g). In a $\mathcal{V}$-PCA $\mathcal{A} = (S, \iota, \mathit{Msg}, \Delta, \mathcal{F})$, the acceptance condition is given by an MSO_t formula $\mathcal{F}$ which may have free variables in $\mathcal{U}$: $\mathsf{Free}(\mathcal{F}) \subseteq \mathcal{U}$. The

transitions are also extended to read bit vectors: $\Delta \subseteq S \times \Sigma \times \mathit{Msg} \times \{0,1\}^{\mathcal{X}} \times S$. The semantics is extended as expected. We use the notation introduced in Section 3. A run is still a labelling $\rho : E \to S$ and if $e \lhd_{\mathsf{msg}} e'$ with $\pi(e) \overset{a\ b}{\longmapsto} \pi(e')$ we require that for some message $m \in \mathit{Msg}$ we have $(\rho^-(e), (a!, m, g(e)), \rho(e)) \in \Delta$ and $(\rho^-(e'), (b?, m, g(e')), \rho(e')) \in \Delta$. The run is accepting if $(P, \longmapsto, \lambda), f \models \mathcal{F}$.

Notice first that we can easily build a $\mathcal{V}$-PCA $\mathcal{A}_{\mathsf{valid}}$ checking that a $\mathcal{V}$-extended MSC (M, f, g) encodes first-order variables faithfully: for each first-order process variable $u \in \mathcal{U}$ there is exactly one process $p \in P$ such that $f(u)(p) = 1$ and similarly for first-order event variables. The automaton for a subformula ψ of φ is the intersection of $\mathcal{A}_{\mathsf{valid}}$ with an automaton depending on ψ and constructed inductively as follows.

- The PCA for the formula $u \overset{a\ b}{\longmapsto} v$ has a single state $S = \{\iota\}$, a single message $\mathit{Msg} = \{m\}$, a full transition table $\Delta = S \times \Sigma \times \mathit{Msg} \times \{0,1\}^{\mathcal{X}} \times S$, and its acceptance condition is simply the formula $\mathcal{F} = u \overset{a\ b}{\longmapsto} v$. This automaton accepts clearly all $\mathcal{V}$-extended MSCs satisfying $u \overset{a\ b}{\longmapsto} v$.
- Another interesting case is the formula $x@u$, since it mixes an event and a process variable. The corresponding $\mathcal{V}$-PCA has two states $S = \{\iota, s\}$ and a single message $\mathit{Msg} = \{m\}$. It is deterministic and complete. It moves from state ι to state s only when reading the (unique) event e associated with x, i.e., such that $g(e)(x) = 1$. The automaton stays in its current state when reading an event e such that $g(e)(x) = 0$. The acceptance condition is $\mathcal{F} = \forall v.(\lambda(v) = s \longleftrightarrow v = u)$.
- For the formula $x \lhd_{\mathsf{msg}} y$ which compares two *events*, we need two states $S = \{\iota, s\}$ and two messages $\mathit{Msg} = \{m, m'\}$. The special message m' is emitted by the event e such that $g(e)(x) = 1$ and can only be received by the event e' such that $g(e')(y) = 1$. When the special message is used, the automaton moves from state ι to state s. All other pairs of events $e \lhd_{\mathsf{msg}} e'$ use message m and keep the state of the automaton unchanged. The acceptance condition $\mathcal{F} = \exists u.(\lambda(u) = s)$ makes sure that the special message has been used.
- Constructing $\mathcal{V}$-PCAs for the formulas $x \lhd_{\mathsf{proc}} y$, $x = y$, and $x \in X$ is easy.
- Let us turn to complementation. Suppose we have a PCA $\mathcal{A}$ for formula $\psi \in \mathrm{MSO}_m$, i.e., $L(\mathcal{A}) \equiv_{(k,ct)} L(\psi)$. By Theorem 9 (which also holds for $\mathcal{V}$-extended MSCs), there is a PCA $\mathcal{A}'$ such that $L(\mathcal{A}') \equiv_{(k,ct)} \mathbb{MSC} \setminus L(\mathcal{A})$. Thus, we have $L(\mathcal{A}') \equiv_{(k,ct)} L(\neg\psi)$.
- Disjunction reduces to "union" of two PCAs and existential (first-order or second-order) event quantification is dealt with via projection of the transition table as usual.
- Existential process quantification is different from event quantification. Suppose we already have a PCA $\mathcal{A}$ for $\psi \in \mathrm{MSO}_m$, say, with acceptance condition $\mathcal{F}$. The PCA for $\exists u.\psi$ is like $\mathcal{A}$, but uses as acceptance condition the formula $\exists u.\mathcal{F}$. Second-order process quantification is handled analogously.